Assignment in Information Retrieval Course

Language Modeling

First of all, to simplify the implementation, we've chosen to gather 50 files from the directory train and test and store them in one list called "set_of_texts".

After collecting 50 files into a list, we begin our process by cleaning the text, which involves replacing any characters that are not letters with a space. Then, we split the text into individual words, using spaces as separators.

Once we've separated the words using white space, our next step is to calculate the frequency of each word. Here's how we do it:

1. We combine all the words into a single list.
2. From these tokens, we generate n-grams, which are groups of adjacent words.
3. We count how often each n-gram occurs.
4. Based on these counts, we calculate the frequency of each n-gram.
5. The results are stored in a DataFrame, with n-grams sorted in descending order of frequency.
6. we return this DataFrame.

- Notice that 1-gram is the term-frequency (TF) so we will use 1-gram to analyze the terms frequency.

After calculating the frequencies, we create two lists: one containing unique words and the other containing all words (with duplicates).

It's worth noting that the word "the" is likely to have the highest frequency at this stage, as we haven't made any changes to the text yet.

	level_0	Count	Frequency
0	the	15616	0.039947
1	of	12447	0.031841
2	and	12301	0.031467
3	in	9299	0.023788
4	to	7212	0.018449

The number of word
output:
The Number of words in the orginal Data is: 390917
The Number of uniqe words in the orginal Data is: 27436

Let's examine whether **Heaps' law** of vocabulary size applies here. According to Heaps' law, if M represents the vocabulary size and T represents the number of words, with $T = 390{,}917$ words in our case, we can investigate the relationship between them.

$$\log_{10} M = 0.49 * \log_{10} T + 1.64 \rightarrow M \approx 44 * \sqrt{T}$$

$$M \approx 44 * \sqrt{390{,}917} = 27{,}510$$

It's evident that the estimated vocabulary size M of **27,510** closely matches the real vocabulary size of **27,436**. This alignment indicates that Heaps' law of vocabulary size holds true in this scenario.

- **Perform a set of linguistic operations**

a. Remove stop words

we'll remove the stopwords because they're not very meaningful.
Here's how we do it:

- We start by getting a list of common English stopwords.
- for each piece of text, we remove these stopwords.
- After cleaning up the text, we count how many words are left and how many unique words there are.

This process helps us focus on the important words for analysis .

After we remove the stopword we got highest frequency is the word gender

	level_0	Count	Frequency
0	gender	4165	0.016491
1	bias	2583	0.010227
2	women	2154	0.008529
3	al	1732	0.006858
4	et	1715	0.006790

and we see that there is a noticeable change in the number of words

The Number of words after removing the Stopwords : 252561
The Number of unique words after removing the Stopwords : 27128

After removing stop words, the count of unique words decreases from 27,436 to 27,128, **indicating the removal of 308 different stop words.** However, the total word count decreases significantly from 390,917 to 252,561, **signifying the removal of 138,356 stop words**. This highlights the prevalence of stop words in the texts.

b. Perform Case Folding

After removing the stopwords, we perform case folding on all the words. simply we convert all the words to lowercase using the `word.lower()` function. After this step, we count the words and the unique words once again.

This ensures consistency in our data and helps us accurately analyze it.

After removing stopwords and performing case folding, the total number of words changes. This is because case folding reduces variations in capitalization, potentially merging words that were previously considered different due to their casing (e.g., "Word" and "word").

output:
The Number of words after Performing Case Folding : 252561
The Number of unique words after Performing Case Folding : 22351

Changing text to lowercase made the number of unique words go down. This happened because it makes all the letters the same size, so words that are the same except for their capitalization are now counted as just one word. So, there are fewer unique words because similar ones with different capitalization are combined into one.
The decrease in the count of unique words from 27,128 to 22,351 suggests that there were **4,813 instances of words that only differed in case** (lowercase or capitalization) within the text.
The most frequency word still the same (their frequency got higher)

:

	level_0	Count	Frequency
0	gender	5176	0.020494
1	bias	2953	0.011692
2	women	2605	0.010314

c. Perform Stemming, using porter

- Stemming: We use a process called stemming to simplify words to their root form. For example, "running" becomes "run." We use a specific algorithm called the Porter stemming algorithm for this.

- Building a Language Model: Next, we create a language model based on these simplified words. It's like building a map of how words are used together in the text.

- Calculating Word Counts: Finally, we count how often each word appears in the text data. This helps us see how stemming affects the overall data.

So, by stemming the text and then analyzing word counts, we can understand better how words are used and grouped together in the text.

After we make all the processes and all the changes, the number of unique words decreases.

output
The Number of words after Performing stemming : 252561
The Number of unique words after Performing stemming : 16325

After performing stemming on the text, words are simplified to their basic form. For instance, "running," "runs," and "ran" are all treated as "run." This simplification reduces the count of unique words because it merges similar variations into a single basic form.

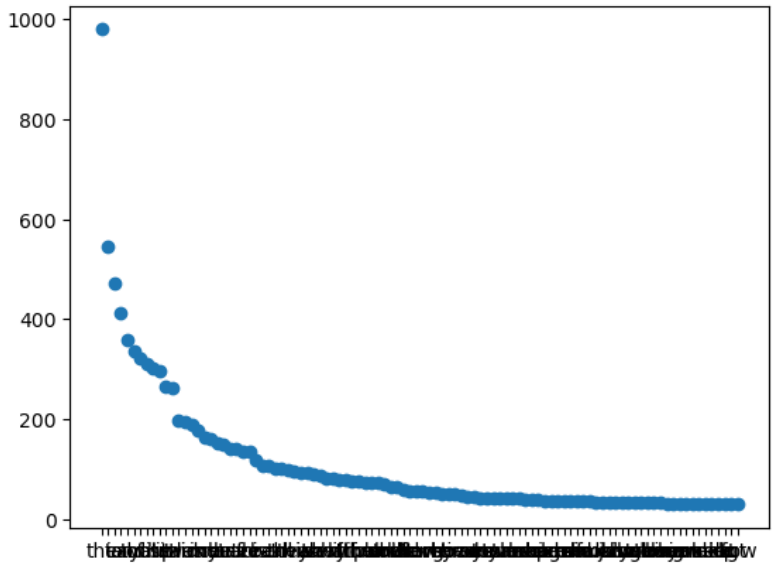
The decrease in the count of unique words from 22,351 to 16,325 suggests that there were **6,026 instances of words that share the same basic form with other words** within the text.

The most frequent words are still the same.
It seems that the stemming algorithm changed the word "bias" to "bia", but it still has the same frequency and didn't change, so that isn't considered a problem.

]:

	level_0	Count	Frequency
0	gender	5522	0.021864
1	bia	2954	0.011696
2	women	2618	0.010366
3	al	1752	0.006937

- Upon plotting the frequency distribution of the 80 most frequent words, it became evident that the distribution closely adheres to **Zipf's law**. Zipf's law states that given a large sample of words used, the frequency of any word is inversely proportional to its rank in the frequency table.



Text classification

In this section, we explore the application of machine learning classifiers to our text data. We use a freely available machine learning library called **scikit-learn**, which provides a wide range of classifiers and tools for machine learning tasks.

We loaded the data from two directories: "gender bias" and "Discrimination discovery". Files from the "gender bias" directory were labeled as 1, while those from the "Discrimination discovery" directory were labeled as 0. This labeling approach enabled the classifiers to differentiate between the content of the documents originating from each directory during the training process.

Performing Linguistic Operations:
Before applying machine learning classifiers, we performed a series of linguistic operations to preprocess the text data. These operations are essential for preparing the data and improving the performance of the classifiers. Here's an overview of the linguistic operations we conducted:

1. Tokenization: We started by tokenizing the text (with function from section 1), which involves splitting it into individual words or tokens. This step allows us to analyze the text at the word level and extract meaningful features for classification .

2. Removing Stopwords: Stopwords are common words that do not carry significant meaning, such as "the," "and," and "is." We removed these stopwords from the text (using function from section 1) to focus on more meaningful words that can help discriminate between different classes.

By performing these linguistic operations, we transformed the raw text data into a more structured and standardized format suitable for machine learning algorithms. This preprocessing step plays a crucial role in improving the accuracy and effectiveness of the classifiers by reducing noise and irrelevant information in the data.

We selected four classifiers for our experiment, each with its own strengths and weaknesses:
1. Multinomial Naive Bayes (MultinomialNB)
2. Logistic Regression
3. K-Nearest Neighbors (KNN)
4. AdaBoost

These classifiers were chosen to assess different types of classification algorithms and their suitability for our text data.

Evaluation:

To evaluate the performance of each classifier, we used a cross-validation technique called Stratified K-Fold. This technique ensures that each fold of the data contains approximately the same proportion of each class, which is important for maintaining the representativeness of the data.

Each classifier was trained and evaluated using 10-fold cross-validation. This means that the data was divided into 10 equally sized folds, and the classifier was trained and tested 10 times, with each fold used once as the test set and the remaining folds used as the training set.

Results:
After evaluating each classifier using cross-validation, we obtained the following accuracy scores:

1. Multinomial Naive Bayes (MultinomialNB) Accuracy: 99.00%
2. Logistic Regression Accuracy: 99.00%
3. K-Nearest Neighbors (KNN) Accuracy: 97.00%
4. AdaBoost Accuracy: 99.00%

Analysis:

Overall, the classifiers performed well on the text classification task, achieving high accuracy scores ranging from 97% to 99%.

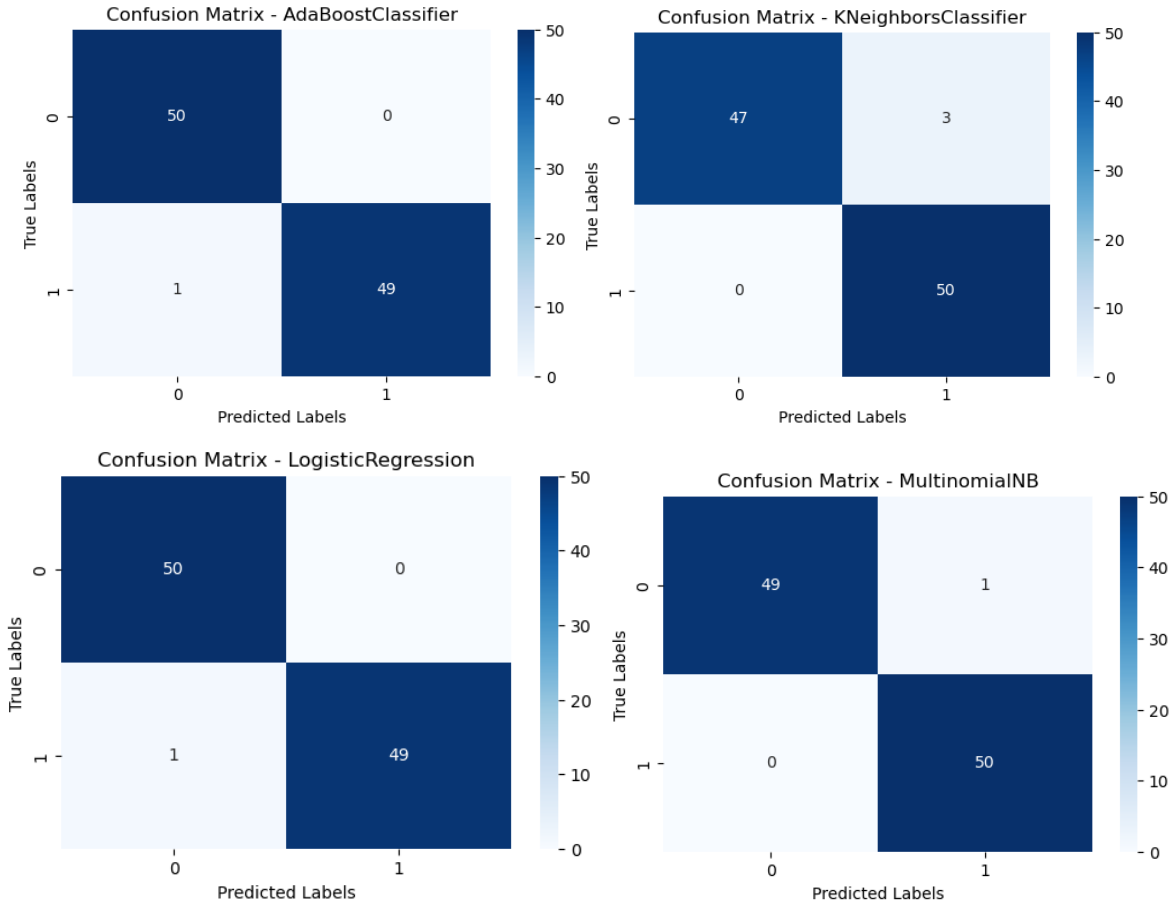
Multinomial Naive Bayes (MultinomialNB), Logistic Regression, and AdaBoost classifiers achieved the highest accuracy scores of 99%. These classifiers are known for their effectiveness in text classification tasks (they are all parametric algorithms), making them suitable choices for our dataset.

K-Nearest Neighbors (KNN) classifier achieved a slightly lower accuracy score of 97%. While KNN is a simple and intuitive algorithm (non-parametric lazy learning), its performance may be affected by the choice of distance metric and the number of neighbors.

In conclusion, our experiment demonstrates the effectiveness of machine learning classifiers in text classification tasks. By selecting appropriate classifiers and using proper evaluation techniques, we can achieve high accuracy in classifying text data.

- The evaluation results:

Confusion Matrix:



The confusion matrix tells us how well our classifiers are doing in identifying different types of data. For Multinomial Naive Bayes and AdaBoost, they got most of the answers right. They correctly identified almost all the positive and negative cases, as seen in the high numbers for true positives (TP) and true negatives (TN). There were only a few mistakes, like saying something was positive when it wasn't (false positive, FP) or saying something wasn't positive when it was (false negative, FN). But those mistakes were very rare, just 1 each.

With Logistic Regression and KNN, things were similar, but not quite as perfect. They still did a good job overall, getting most of the answers right. However, there were a couple more mistakes. Logistic Regression said one thing was negative when it was actually positive, and KNN said three things were positive when they weren't. So, while they mostly got it right, there were a few slip-ups along the way.

Overall, our classifiers did a great job in classifying the data into the right categories. They got most of the answers correct, with only a few errors.

Precision, Recall and F1 :

	Precision	Recall	F1 Score
MultinomialNB	0.983333	1.00	0.990909
LogisticRegression	1.000000	0.98	0.988889
KNeighborsClassifier	0.950000	1.00	0.972727
AdaBoostClassifier	1.000000	0.98	0.988889

The precision, recall, and F1 scores for all classifiers are nearly identical, showing consistent and strong performance. This means each classifier excelled in making accurate predictions.
Their close similarity indicates that all models were well-trained and dependable in categorizing the data accurately, with scores close to one.

The Multinomial Naive Bayes classifier achieved high precision and recall, indicating that it made accurate positive predictions.
The Logistic Regression and AdaBoost classifiers also performed well, with high precision and recall scores.
However, the K-Nearest Neighbors classifier had slightly lower precision, suggesting that it made more false positive predictions, while still maintaining a high recall rate.

These results complement the accuracy scores obtained earlier, providing a more comprehensive understanding of the classifiers' performance. Despite some variations in precision and recall, all classifiers demonstrated strong overall performance in accurately classifying the data.

Text Clustering:
In this section, we explore text clustering, a fundamental task in natural language processing aimed at grouping similar documents together. We follow these steps:

1.Data Collection and Preprocessing:
We collect documents from these four directories: train and test set, Discrimination discovery data, formal fairness data and explainable AI data. After collecting the data, we perform text preprocessing steps, including tokenization and stop word removal.

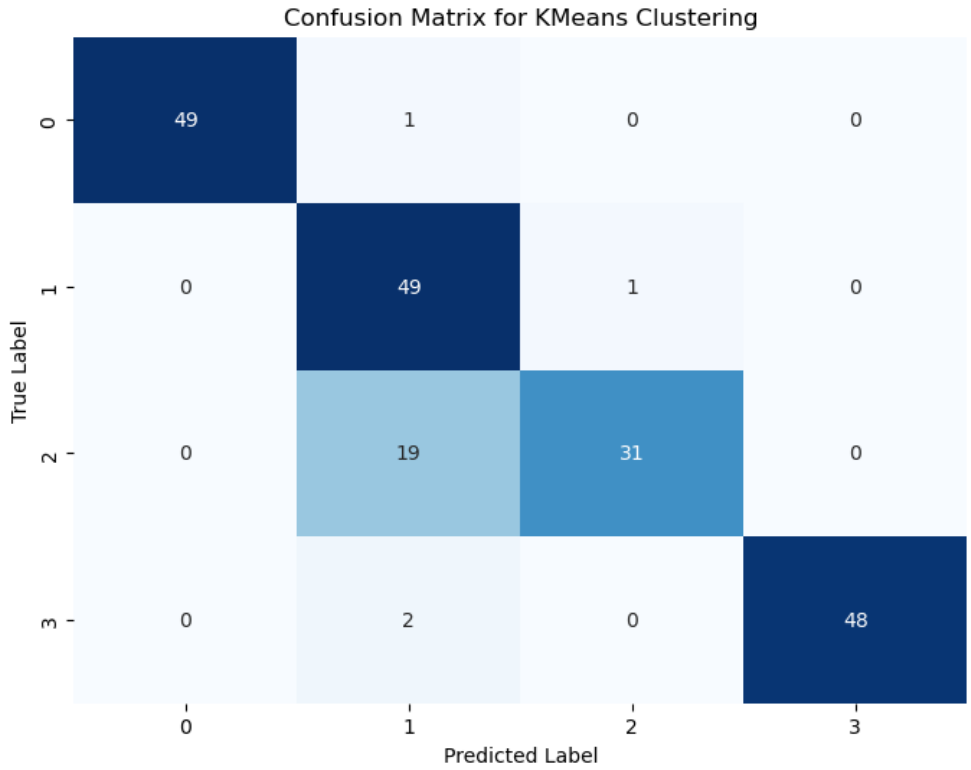
2.Labeling and Concatenation:
To facilitate clustering, we label the documents from each directory with distinct numeric labels. We assign label 0 to the documents from the training and test set, label 1 to the formal fairness data, label 2 to the Discrimination discovery, and label 3 to the explainable AI data. Next, we concatenate all the labeled data frames into a single combined data frame.

3.Clustering with K-Means:
We employ the K-Means clustering algorithm to group the documents into four clusters. Before applying K-Means, we transform the text data into TF-IDF features using the TF-IDF vectorizer. Then, we initialize and fit the K-Means algorithm to the TF-IDF features. After clustering, we add cluster labels to the combined data frame.

4. Evaluation
To measure the effectiveness of our clustering approach, we calculated the clustering accuracy by comparing the assigned cluster labels with the original labels. This metric signifies the percentage of accurately clustered documents relative to the total.

Our clustering accuracy reached **88.50%**, indicating a high level of success in grouping similar documents. This result demonstrates the robustness and effectiveness of the K-means clustering algorithm in our text clustering task. However, further investigation into misclassified documents could offer insights for potential enhancements in clustering performance.

Confusion Matrix:



In our data, we noticed that many files labeled as 2(Discrimination discovery) ended up in cluster 1(formal fairness data).
This mismatch suggests that the clustering algorithm struggled to distinguish between the characteristics of these categories.

For example, if label 1 and label 2 are supposed to represent different topics, the frequent misclassification indicates significant similarity between them. This could be because they share common keywords, themes, or language patterns
This is possible while label 2 is for the "Discrimination discovery data", which is closely related to the topic of the directory labeled 1, which is about "formal fairness data." Therefore, they may share some keywords or themes that confuse the clustering algorithm and cause misclassification.

Conclusion:
In conclusion, the K-means algorithm effectively clustered the majority of labels 0, 1, and 3 accurately, indicating its robust performance in grouping similar documents. However, the misclassification observed between labels 1 and 2 suggests that sometimes, misclassifications can occur when dealing with related topics that share common keywords and themes. This underscores the importance of meticulous data curation and labeling to enhance the accuracy of clustering algorithms in categorizing textual data effectively.